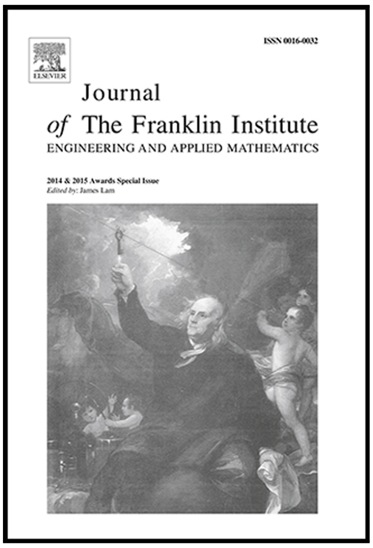

Integer Codes Correcting Burst and Random Asymmetric Errors within a Byte

Aleksandar Radonjic , Vladimir Vujicic

PII: S0016-0032(17)30622-1
DOI: 10.1016/j.jfranklin.2017.11.033
Reference: FI 3243

To appear in: *Journal of the Franklin Institute*

Received date: 23 February 2016
Revised date: 6 May 2017
Accepted date: 26 November 2017

Integer Codes Correcting Burst and Random Asymmetric Errors within a Byte

Aleksandar Radonjic and Vladimir Vujicic

***Abstract*-In most communication networks, error probabilities 1→ 0 and 0 → 1 are equally likely to occur. However, in some optical networks, such as local and access networks, this is not the case. In these networks, the number of received photons never exceeds the number of transmitted ones. Hence, if the receiver operates correctly, only 1 → 0 errors can occur. Motivated by this fact, in this paper, we present a class of integer codes capable of correcting burst and random asymmetric (1 → 0) errors within a *b*-bit byte. Unlike classical codes, the proposed codes are defined over the ring of integers modulo 2^b – 1. As a result, they have the potential to be implemented in software without any hardware assist.**

***Keywords*: Integer codes, error correction, asymmetric errors, optical networks.**

1. Introduction

Most classes of codes have been designed for use on symmetric channels, where 0 → 1 and 1 → 0 errors occur with equal probability. In some practical systems, however, the probability of 0 → 1 errors is extremely small. One such example are optical networks without optical amplifiers (ONWOAs) (e.g. local and access networks) [1]. In these networks, the number of received photons never exceed the number of sent ones [2], [3]. Hence, if the receiver operates correctly, only asymmetric (1 → 0) errors may occur. These include random errors as well as bursts of length up to 8 bits [4], [5], [6], [7].

Although this fact is known for decades, there has been very little work on codes for ONWOAs. Some classes of burst asymmetric codes were presented in [8], [9], but without any information regarding their implementation. The same holds for [10], [11], where the authors proposed two classes of codes capable of correcting random and byte asymmetric errors, respectively. Unlike the above-mentioned works, in [13] we have presented a class of codes that is specially designed for use in ONWOAs. The main advantage of these codes is the ability to exploit high computing power of the network nodes (PCs, servers, routers, switches, OLT/ONU units, etc.) [1]. This has been achieved by using integer and lookup table (LUT) operations, which are supported by all processors. Unfortunately, in terms of error correction, these codes were quite weaker than [8], [9], [10], [11]. More precisely, they were able to correct spotty byte asymmetric errors, unlike [8], [9], [10], [11] which are designed to correct multiple asymmetric errors within a byte or codeword.

In this paper, we present a class of codes that significantly outperforms the codes proposed in [13]. Unlike them, the codes presented in this paper can correct two types of errors within a b-bit byte: burst asymmetric errors up to l ($< b$) bits (l/b BA errors) and random asymmetric errors up to t ($< l$) bits (t/b RA errors). The only penalty for this improved performance is a slight increase in memory requirements. On the other hand, the proposed codes retain all the desirable properties of [13] including systematic structure, simple encoding/decoding algorithm and fast error control procedure based on table lookups.

The organization of this paper is as follows: Section 2 deals with the construction of integer codes capable of correcting l/b BA and t/b RA errors (integer $(\mathrm{B}_l\mathrm{AEC/R}_t\mathrm{AEC})_b$ codes). The implementation strategy and theoretical decoding throughputs for these codes are described and evaluated in Section 3. In Section 4 the proposed codes are compared with the existing codes of similar properties. Finally, Section 5 concludes the paper. For ease of reading, a list of notations is given in Table 1.

Table 1. Notations Used in This Paper.

Symbol	Meaning
B_i	Integer value of the *i*-th *b*-bit data byte at the sender side
C_{B}	Integer value of the *b*-bit check-byte at the sender side
$\hat{B}_i$	Integer value of the received *i*-th *b*-bit data byte
$\hat{C}_{\mathrm{B}}$	Integer value of the received *b*-bit check-byte
$C_{\hat{\mathrm{B}}}$	Integer value of the *b*-bit check-byte at the receiver side

2. Codes Construction

A. Encoding and Decoding Procedures

Let Z_{2^b-1} = {0, 1,…, 2^b - 2} be the ring of integers modulo 2^b - 1, and let $C_i \in Z_{2^b-1} \setminus \{0,1\}$, where $1 \le i \le k$. In addition, let us assume that the encoder/decoder uses a LUT_1 to store the coefficients $C_i \in Z_{2^b-1} \setminus \{0,1\}$. In that case, the encoder will generate the check-byte C_{B} by using the following operations:

$$C_{\mathrm{B}} = [C_1 \cdot B_1 + \cdots + C_k \cdot B_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot B_i \ (\mathrm{mod}\ 2^b - 1) \tag{1}$$

At the receiver, the decoder will perform the same calculation

$$C_{\hat{\mathrm{B}}} = [C_1 \cdot \hat{B}_1 + \cdots + C_k \cdot \hat{B}_k] \ (\mathrm{mod}\ 2^b - 1) = \sum_{i=1}^{k} C_i \cdot \hat{B}_i \ (\mathrm{mod}\ 2^b - 1) \tag{2}$$

after which the syndrome S will be formed

$$S = [C_{\hat{\mathrm{B}}} - \hat{C}_{\mathrm{B}}] \ (\mathrm{mod}\ 2^b - 1) \tag{3}$$

Obviously, when $S \neq 0$, the codeword is corrupted by one or more errors.

B. *Necessary and Sufficient Conditions*

In order to be able to correct all *l*/*b* BA and *t*/*b* RA errors, the decoder must know which nonzero values of *S* correspond to these errors. However, before we move to the mathematical details, let us observe that *t*/*b* RA errors spaced less than *l* bits (short *t*/*b* RA errors) can be treated as *l*/*b* BA errors. Hence, in addition to *l*/*b* BA errors, we need to consider only *t*/*b* RA errors that are spaced *d* ($l < d < b$) bits apart (long *t*/*b* RA errors) (Fig. 1).

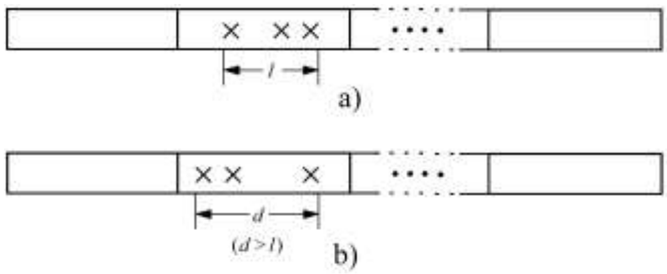

Fig. 1. (a) Short *t*/*b* RA errors and (b) long *t*/*b* RA errors.

Now we can give two definitions, the first of which stems from our previous work [12].

Definition 1. *Let* $e_1 = \{1\}$ and $e_w = \{2^{w-1}+1, 2^{w-1}+3, \ldots, 2^w - 1\}$ *be the sets of odd integers, and let l be an integer such that* $2 \le w \le l < b$. *Then, the set of syndromes corresponding to l/b BA errors is defined as*

$$\varepsilon_1 = s_1 \cup s_2 \tag{4}$$

where

$$s_1 = \left\{ \bigcup_{m=1}^{l} \left(2^r \cdot e_m\right) \left(\bmod 2^b - 1\right):\ 0 \le r \le b - m \right\} \tag{5}$$

$$s_2 = \left\{ \bigcup_{m=1}^{l} \bigcup_{i=1}^{k} \left(-C_i \cdot 2^r \cdot e_m\right) \left(\bmod 2^b - 1\right):\ 0 \le r \le b - m \right\} \tag{6}$$

Definition 2. *Let* $f_2 = \{2^s + 2^z\}$ and $f_v = \{2^s + 2^{x_1} + 2^{x_2} + \cdots + 2^{x_{v-2}} + 2^z\}$ *be the sets of integers, where* $3 \le v < l < b$, $0 \le s < x_1 < x_2 < \cdots < x_{v-2} < z$ and $s + l \le z \le b - 1$. *Then, the set of syndromes corresponding to long t/b RA errors is defined as*

$$\varepsilon_2 = s_3 \cup s_4 \tag{7}$$

where

$$s_3 = \left\{ \bigcup_{n=2}^{t} \left(f_n\right) \left(\bmod 2^b - 1\right):\ 2 \le t < l \right\} \tag{8}$$

$$s_4 = \left\{ \bigcup_{n=2}^{t} \bigcup_{i=1}^{k} \left(-C_i \cdot f_n\right) \left(\bmod 2^b - 1\right):\ 2 \le t < l \right\} \tag{9}$$

Although the expressions (1)-(9) provide a theoretical basis for the construction of (*kb* + *b*, *kb*) integer (B_lAEC/R_tAEC)$_b$ codes, they do not give the explicit information about the number of nonzero syndromes. Hence, we need the following theorem.

Theorem 1. *The codes defined by (1)-(3) can correct all l/b BA and t/b RA errors iff there exist k mutually different coefficients* $C_i \in Z_{2^b-1} \setminus \{0,1\}$ *such that*

1. $|\varepsilon_1| = (k+1)\cdot\left[2^{l-1}\cdot(b-l+2)-1\right]$
2. $|\varepsilon_2| = (k+1)\cdot\sum_{i=0}^{b-l-1}\sum_{j=2}^{t}(b-l-i)\cdot\binom{l+i-1}{j-2}$
3. $\varepsilon_1 \cap \varepsilon_2 = \varnothing$

where $|\mathrm{A}|$ denotes the cardinality of A, and $\mathrm{A}\cap\mathrm{B}$ the intersection of A and B.

Proof. Condition 1 of this theorem says that *l*/*b* BA errors generate $(k+1)\cdot\left[2^{l-1}\cdot(b-l+2)-1\right]$ syndromes that are nonzero. To prove this, observe that the set s_1 can be expressed as

$$s_1 = \bigcup_{m=1}^{l} a_m$$

where

$$a_1 = \left\{(1)\cdot 2^r \ (\mathrm{mod}\ 2^b-1):\ \ r=0,1,\ldots,b-1\right\}$$
$$a_2 = \left\{(3)\cdot 2^r \ (\mathrm{mod}\ 2^b-1):\ \ r=0,1,\ldots,b-2\right\}$$
$$\vdots$$
$$a_l = \left\{(2^{l-1}+1,\ 2^{l-1}+3,\ 2^{l-1}+5,\ldots,2^l-1)\cdot 2^r \ (\mathrm{mod}\ 2^b-1):\ \ r=0,1,\ldots,b-l\right\}$$

Consequently, we can write

$$|s_1| = \sum_{m=1}^{l}|a_m| = b + \sum_{h=2}^{l}2^{h-2}\cdot(b-h+1).$$

To calculate the sum we can use equalities

$$\sum_{j=1}^{l-1}x^j = \frac{x^l-1}{x-1}-1$$
$$\sum_{j=1}^{l-1}j\cdot x^j = x\cdot\frac{1-x^{l-1}}{(1-x)^2}-\frac{(l-1)\cdot x^l}{1-x}$$

wherefrom it follows that

$$|s_1| = b+(b+1)\cdot(2^{l-1}-1) - l\cdot 2^{l-1}+2^{l-1} = 2^{l-1}\cdot(b-l+2)-1.$$

On the other hand, from (5) and (6) we see that the set s_2 can be expressed as

$$s_2 = \bigcup_{i=1}^{k} b_i$$

where

$$b_1 = \left\{ \left(-C_1 \cdot s_1 \right) \left(\bmod 2^b - 1 \right) \right\}$$
$$b_2 = \left\{ \left(-C_2 \cdot s_1 \right) \left(\bmod 2^b - 1 \right) \right\}$$
$$\vdots$$
$$b_k = \left\{ \left(-C_k \cdot s_1 \right) \left(\bmod 2^b - 1 \right) \right\}$$

From this it is easy to conclude that the syndromes caused by l/b BA errors will be different and not equal to zero iff there exist k mutually different coefficients $C_i \in Z_{2^b-1} \setminus \{0, 1\}$ such that

$$s_1 \cap s_2 = s_1 \cap b_1 \cap b_2 \cap \cdots \cap b_k = \varnothing$$
$$|s_1| = |b_1| = |b_2| = \cdots = |b_k| = 2^{l-1} \cdot (b - l + 2) - 1.$$

In that and only in that case, the set ε_1 will have

$$|\varepsilon_1| = |s_1| + |s_2| = |s_1| + k \cdot |s_1| = (k+1) \cdot \left[2^{l-1} \cdot (b - l + 2) - 1 \right]$$

nonzero elements. In a similar way Condition 2 says that long t/b RA errors generate $(k+1) \cdot \sum_{i=0}^{b-l-1} \sum_{j=2}^{t} (b-l-i) \cdot \binom{l+i-1}{j-2}$ syndromes that are nonzero. To prove this, note that the set s_3 can be expressed as

$$s_3 = \bigcup_{u=0}^{b-l-1} c_u$$

where

$$c_0 = \left\{ \left(2^s + \underbrace{2^e + 2^f + \cdots + 2^p}_{t-2 \text{ addends}} + 2^{s+l} \right) (\bmod 2^b - 1) : \ s = 0, 1, \ldots, b - l - 1 \right\}$$
$$c_1 = \left\{ \left(2^s + \underbrace{2^e + 2^f + \cdots + 2^p}_{t-2 \text{ addends}} + 2^{s+l+1} \right) (\bmod 2^b - 1) : \ s = 0, 1, \ldots, b - l - 2 \right\}$$
$$\vdots$$
$$c_{b-l-1} = \left\{ \left(2^s + \underbrace{2^e + 2^f + \cdots + 2^p}_{t-2 \text{ addends}} + 2^{s+b-1} \right) (\bmod 2^b - 1) : \ s = 0 \right\}$$

and $s < e < f < \cdots < p < s + l + u$ for any $u = 0, 1, \ldots, b - l - 1$. Consequently, we can write

$$|c_0| = (b - l) \cdot \binom{l-1}{t-2}$$
$$|c_1| = (b - l - 1) \cdot \binom{l}{t-2}$$
$$\vdots$$
$$|c_{b-l-1}| = 1 \cdot \binom{b-2}{t-2}$$

wherefrom it follows that

$$\left|s_3\right| = \sum_{u=0}^{b-l-1}\left|c_u\right| = \sum_{i=0}^{b-l-1}\sum_{j=2}^{t}\left(b-l-i\right)\cdot\binom{l+i-1}{j-2}.$$

On the other hand, from (8) and (9) we see that the set s_4 can be expressed as

$$s_4 = \bigcup_{i=1}^{k} d_i$$

where

$$d_1 = \left\{\left(-C_1\cdot s_3\right)\left(\mathrm{mod}\ 2^b-1\right)\right\}$$
$$d_2 = \left\{\left(-C_2\cdot s_3\right)\left(\mathrm{mod}\ 2^b-1\right)\right\}$$
$$\vdots$$
$$d_k = \left\{\left(-C_k\cdot s_3\right)\left(\mathrm{mod}\ 2^b-1\right)\right\}$$

Obviously, the syndromes caused by long t/b RA errors will be different and not equal to zero iff there exist k mutually different coefficients $C_i \in Z_{2^b-1}\setminus\{0,1\}$ such that

$$s_3 \cap s_4 = s_3 \cap d_1 \cap d_2 \cap \cdots \cap d_k = \varnothing$$
$$\left|s_3\right| = \left|d_1\right| = \left|d_2\right| = \cdots = \left|d_k\right| = \sum_{i=0}^{b-l-1}\sum_{j=2}^{t}\left(b-l-i\right)\cdot\binom{l+i-1}{j-2}$$

Only in that case the set ε_2 will have

$$\left|\varepsilon_2\right| = \left|s_3\right| + \left|s_4\right| = \left|s_3\right| + k\cdot\left|s_3\right| = (k+1)\cdot\sum_{i=0}^{b-l-1}\sum_{j=2}^{t}\left(b-l-i\right)\cdot\binom{l+i-1}{j-2}$$

nonzero elements. Finally, Condition 3 is a necessary condition for distinguishing l/b BA errors from long t/b RA errors. So, (kb + b, kb) integer $(\mathrm{B}_l\mathrm{AEC}/\mathrm{R}_t\mathrm{AEC})_b$ codes must satisfy all the conditions 1 to 3. Conversely, if the codes satisfy conditions 1 to 3, then we can distinguish l/b BA errors from long t/b RA errors. We can also correct all l/b BA and t/b RA errors. Therefore, these codes are (kb + b, kb) integer $(\mathrm{B}_l\mathrm{AEC}/\mathrm{R}_t\mathrm{AEC})_b$ codes. □

From a practical point of view it is sufficient to use the codes capable of correcting up to three RA errors within a b-bit byte. Such a choice is completely in agreement with the theory [6] as well as with experimental results [4], [5], [6], [7]. In addition, it significantly reduces the size of the syndrome table which is used during the error correction process.

Theorem 2. *Let t = 2, 3 and let* $\xi_{l,t,b,k} = \varepsilon_1 \cup \varepsilon_2$ *be the error set for (kb + b, kb) integer* $(B_lAEC/R_tAEC)_b$ *codes. Then,*

$$\left|\xi_{l,2,b,k}\right| = \left|\varepsilon_1\right| + \left|\varepsilon_2\right| = \left[2^{l-1}\cdot(b-l+2) - 1 + \frac{(b-l)^2 + b - l}{2}\right]\cdot\left(k+1\right)$$
$$\left|\xi_{l,3,b,k}\right| = \left|\varepsilon_1\right| + \left|\varepsilon_2\right| = \left[2^{l-1}\cdot(b-l+2) - 1 + \frac{(b-l)^2 + b - l}{2}\cdot b\right]\cdot\left(k+1\right) - \left[\frac{2\cdot(b-l)^3 + 3\cdot(b-l)^2 + b - l}{6}\right]\cdot\left(k+1\right).$$

Proof. This theorem follows directly from Theorem 1.

To illustrate the applicability of Theorem 1, we show results of a computer-search for the codes with parameters b = 32, $2 \le t \le 3$ and $8 \le l \le 9$ (Appendix A). The software used for the search is written in MATLAB (Appendix B). Its aim is to find the first 128 coefficients $C_i \in Z_{2^{32}-1} \setminus \{0,1\}$ that satisfy the conditions of Theorem 1.

C. Error Control Procedure

Suppose that the decoder uses the syndrome table (LUT_2) with $\left|\xi_{l,t,b,k}\right|$ entries, where each entry has $2b+\lceil log_2(k+1)\rceil$ bits. Unlike the LUT_1, which stores the coefficients C_i, this table is generated using (4)-(9). Its aim is to describe the relationship between the nonzero syndrome (element of the set $\xi_{l,t,b,k}$), error location (i) and error vector (e) (Fig. 2).

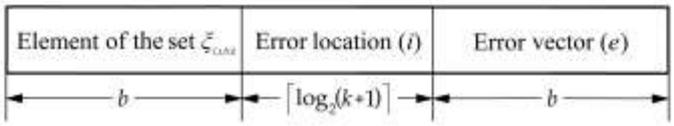

Fig. 2. Bit-width of one LUT_2 entry.

Bearing this in mind, it is easy to see that the main task of the decoder is to find the entry where the first b bits match that of the syndrome S. For that purpose, the decoder must perform a series of table lookups, where each lookup includes a comparison of two b-bit vectors (similar to routing [14]). In the end, after n_{TL} table lookups, the decoder will declare failure $(S \notin \xi_{l,t,b,k})$ or execute $(S \in \xi_{l,t,b,k})$ one of the following operations:

- *for l/b BA errors within the check-byte*

$$C_B = [\hat{C}_B + e]\ (\text{mod}\ 2^b - 1); \qquad (10)$$
$$e = [2^r \cdot e_m]\ (\text{mod}\ 2^b - 1),\ 0 \le r \le b-m,\ 1 \le m \le l < b;$$

- *for l/b BA errors within the i-th data byte*

$$B_i = [\hat{B}_i + e]\ (\text{mod}\ 2^b - 1),\ 1 \le i \le k; \qquad (11)$$
$$e = [2^r \cdot e_m]\ (\text{mod}\ 2^b - 1),\ 0 \le r \le b-m,\ 1 \le m \le l < b;$$

- *for t/b RA errors within the check-byte*

$$C_B = [\hat{C}_B + e]\ (\text{mod}\ 2^b - 1); \qquad (12)$$
$$e = [f_n]\ (\text{mod}\ 2^b - 1),\ 2 \le n \le t < b;$$

- *for t/b RA errors within the i-th data byte*

$$B_i = [\hat{B}_i + e]\ (\text{mod}\ 2^b - 1),\ 1 \le i \le k; \qquad (13)$$
$$e = [f_n]\ (\text{mod}\ 2^b - 1),\ 2 \le n \le t < b;$$

Although error correction procedure is very simple, it is clear that its efficiency depends on the number of compares. For this reason it is very important that the elements of $\xi_{l,t,b,k}$ are sorted in increasing order. In that case it is possible to use binary search algorithm, which requires n_{TL} table lookups $(1 \le n_{\mathrm{TL}} \le \lfloor log_2 |\xi_{l,t,b,k}| \rfloor + 2)$ [15].

Example 1. Let b =10, t = 2, l = 3, k = 2, C_1= 3 and C_2 = 13. According to Lemma 1, the LUT_2 will have $|\xi_{3,2,10,2}| = 189$ entries (Table 2). Given this, let us assume that we want to transmit 20 bits of data, D = 1011100000 0111010011. In that case, after calculating

$$C_{\mathrm{B}} = [C_1 \cdot B_1 + C_2 \cdot B_2] \;(\mathrm{mod}\; 2^b - 1) = [2208 + 6071] \;(\mathrm{mod}\, 1023) = 95$$

the codeword C_{w} = 1011100000 0111010011 0001011111 will have 30 bits.

Table 2. The Syndrome Table (LUT_2) for (30, 20) Integer ($\mathrm{B_3AEC/R_2AEC})_{10}$ Decoder.

	Element of the set $\xi_{3,2,10,2}$	i	e		*Element of the set* $\xi_{3,2,10,2}$	i	e		*Element of the set* $\xi_{3,2,10,2}$	i	e		*Element of the set* $\xi_{3,2,10,2}$	i	e
1	1	3	1	**49**	165	2	66	**97**	507	1	513	**145**	828	1	65
2	2	3	2	**50**	174	2	144	**98**	510	1	512	**146**	831	1	64
3	3	3	3	**51**	178	2	65	**99**	512	3	512	**147**	841	2	14
4	4	3	4	**52**	191	2	64	**100**	513	3	513	**148**	855	1	56
5	5	3	5	**53**	192	3	192	**101**	514	3	514	**149**	867	2	12
6	6	3	6	**54**	207	1	272	**102**	516	3	516	**150**	879	1	48
7	7	3	7	**55**	224	3	224	**103**	520	3	520	**151**	887	2	640
8	8	3	8	**56**	231	1	264	**104**	528	3	528	**152**	893	2	10
9	9	3	9	**57**	243	1	260	**105**	543	1	160	**153**	894	1	384
10	10	3	10	**58**	246	2	768	**106**	544	3	544	**154**	896	3	896
11	12	3	12	**59**	249	1	258	**107**	555	2	36	**155**	903	1	40
12	14	3	14	**60**	252	1	257	**108**	556	2	272	**156**	906	2	9
13	16	3	16	**61**	255	1	256	**109**	573	2	192	**157**	915	1	36
14	17	3	17	**62**	256	3	256	**110**	576	3	576	**158**	919	2	8
15	18	3	18	**63**	257	3	257	**111**	581	2	34	**159**	921	1	34
16	20	3	20	**64**	258	3	258	**112**	590	2	112	**160**	924	1	33
17	24	3	24	**65**	260	3	260	**113**	591	1	144	**161**	927	1	32
18	28	3	28	**66**	264	3	264	**114**	594	2	33	**162**	932	2	7
19	32	3	32	**67**	272	3	272	**115**	607	2	32	**163**	939	1	28
20	33	3	33	**68**	278	2	136	**116**	615	1	136	**164**	945	2	6
21	34	3	34	**69**	288	3	278	**117**	627	1	132	**165**	951	1	24
22	36	3	36	**70**	295	2	56	**118**	628	2	896	**166**	955	2	320
23	40	3	40	**71**	297	2	528	**119**	633	1	130	**167**	958	2	5
24	48	3	48	**72**	314	2	448	**120**	636	1	129	**168**	963	1	20
25	56	3	56	**73**	318	1	576	**121**	639	1	128	**169**	969	1	18
26	63	1	320	**74**	320	3	320	**122**	640	3	640	**170**	971	2	4
27	64	3	64	**75**	330	2	132	**123**	659	2	28	**171**	972	1	17
28	65	3	65	**76**	348	2	288	**124**	660	2	264	**172**	975	1	16
29	66	3	66	**77**	351	1	224	**125**	687	1	112	**173**	981	1	14
30	68	3	68	**78**	356	2	130	**126**	696	2	576	**174**	984	2	3
31	72	3	72	**79**	369	2	129	**127**	702	1	448	**175**	987	1	12
32	80	3	80	**80**	381	1	896	**128**	711	2	24	**176**	989	2	160
33	87	2	72	**81**	382	2	128	**129**	712	2	260	**177**	993	1	10
34	89	2	544	**82**	384	3	384	**130**	735	1	96	**178**	996	1	9
35	96	3	96	**83**	399	2	48	**131**	738	2	258	**179**	997	2	2
36	112	3	112	**84**	401	2	520	**132**	751	2	257	**180**	999	1	8
37	123	2	384	**85**	414	1	544	**133**	763	2	20	**181**	1002	1	7
38	126	1	640	**86**	447	1	192	**134**	764	2	256	**182**	1005	1	6
39	128	3	128	**87**	448	3	448	**135**	765	1	768	**183**	1006	2	80
40	129	3	129	**88**	453	2	516	**136**	768	3	768	**184**	1008	1	5
41	130	3	130	**89**	462	1	528	**137**	783	1	80	**185**	1010	2	1
42	132	3	132	**90**	479	2	514	**138**	789	2	18	**186**	1011	1	4
43	136	3	136	**91**	486	1	520	**139**	798	2	96	**187**	1014	1	3
44	139	2	68	**92**	492	2	513	**140**	802	2	17	**188**	1017	1	2
45	144	3	144	**93**	498	1	516	**141**	807	1	72	**189**	1020	1	1
46	157	2	224	**94**	503	2	40	**142**	815	2	16				
47	159	1	288	**95**	504	1	514	**143**	819	1	68				
48	160	3	160	**96**	505	2	512	**144**	825	1	66				

Scenario 1 (*l/b* BA error): Let us assume that during data transmission an error on the 3rd, 4th and 5th bit has occurred ($\hat{C}_{\mathrm{w}}$ = 10<u>000</u>00000 0111010011 0001011111). After calculating

$$C_{\hat{\mathrm{B}}} = [C_1 \cdot \hat{B}_1 + C_2 \cdot \hat{B}_2] \pmod{2^b - 1} = [1536 + 6071] \pmod{1023} = 446$$

$$S = [C_{\hat{\mathrm{B}}} - \hat{C}_{\mathrm{B}}] \pmod{2^b - 1} = [446 - 95] \pmod{1023} = 351$$

the decoder will check whether the value S = 351 belongs to the set $\xi_{3,2,10,2}$ (Table 2). After completing this task, it will perform error correction by using

$$B_1 = [\hat{B}_1 + e] \pmod{2^b - 1} = [512 + 224] \pmod{1023} = 736.$$

Scenario 2 (*t/b* RA error): Suppose that during data transmission an error on the 12th and 19th bit has occurred ($\hat{C}_{\mathrm{w}}$ =1011100000 0<u>0</u>1101000<u>0</u>1 0001011111). In that case, after calculating

$$C_{\hat{\mathrm{B}}} = [C_1 \cdot \hat{B}_1 + C_2 \cdot \hat{B}_2] \pmod{2^b - 1} = [2208 + 2717] \pmod{1023} = 833$$

$$S = [C_{\hat{\mathrm{B}}} - \hat{C}_{\mathrm{B}}] \pmod{2^b - 1} = [833 - 95] \pmod{1023} = 738$$

the decoder will conclude that the value S = 738 indicates an error within the second byte (Table 2). On the basis of this information it will perform the error correcting procedure by using

$$B_2 = [\hat{B}_2 + e] \pmod{2^b - 1} = [209 + 258] \pmod{1023} = 467.$$

3. Implementation Strategy and Theoretical Decoding Throughputs

An important feature of ONWOAs, which is already mentioned, refers to the architecture of the network nodes. Namely, regardless of its role, each network node (PC, server, router, switch, ONU unit, etc.) [1] is equipped with a processor (general purpose or network processor) that is optimized for integer and LUT operations [16], [17]. Thus, it is interesting to investigate which of the proposed codes have the potential to be implemented on existing network hardware.

Without loss of generality, let us suppose that all network nodes are equipped with quad-core processors (Fig. 3) having the following specifications [17], [18]:

1) clock rate: $C_R = 3.5 \cdot 10^9$ Hz,
2) integer addition/subtraction operation: 1 cycle latency,
3) integer multiplication operation: 3 cycles latency,
4) 128-bit shift operation: 1 cycle latency,
5) modulo reduction operation: 1 cycle latency,
6) comparison operation: 1 cycle latency,
7) access to the L1 cache (32 KB per core): 4 cycles latency,
8) access to the L2 cache (256 KB per core): 11 cycles latency,
9) access to the L3 cache (16 MB shared): 28 cycles latency.

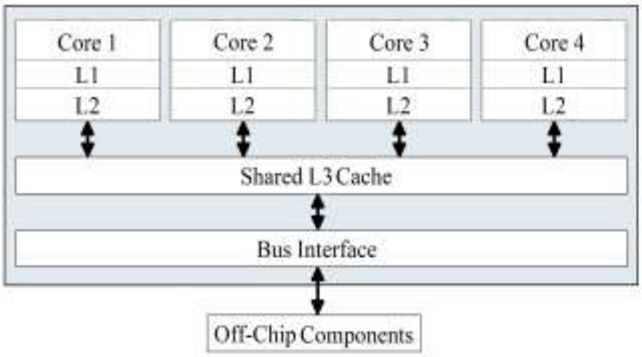

Fig. 3. Block diagram of a quad-core processor.

Table 3. Memory Requirements for Some $(B_l AEC/R_3 AEC)_{32}$ Codes.

Code	l	Memory Requirements for Storing the Coefficients C_i	Memory Requirements for Storing the Syndrome Table	# of Table Lookups
(1056, 1024)	8	4 x 128 B	2.32 MB	$1 \le n_{TL} \le 20$
(1056, 1024)	9	4 x 128 B	3.15 MB	$1 \le n_{TL} \le 20$
(2080, 2048)	8	4 x 256 B	4.63 MB	$1 \le n_{TL} \le 20$
(2080, 2048)	9	4 x 256 B	6.29 MB	$1 \le n_{TL} \le 21$
(4128, 4096)	8	4 x 512 B	9.32 MB	$1 \le n_{TL} \le 21$
(4128, 4096)	9	4 x 512 B	12.66 MB	$1 \le n_{TL} \le 22$

In addition, let us suppose that the coefficients C_i are stored in each of the four L1 caches and that the syndrome table is placed into the L3 cache (Table 3). In that case, the processing cores will perform the following operations:

- *Core 1*

$$C_{\hat{B}1} = [C_1 \cdot \hat{B}_1 + C_2 \cdot \hat{B}_5 + \cdots + C_k \cdot \hat{B}_{4\cdot(k-1)+1}] \pmod{2^b - 1} = \sum_{i=1}^{k} C_i \cdot \hat{B}_{4\cdot(i-1)+1} \pmod{2^b - 1} \tag{14}$$

- *Core 2*

$$C_{\hat{B}2} = [C_1 \cdot \hat{B}_2 + C_2 \cdot \hat{B}_6 + \cdots + C_k \cdot \hat{B}_{4\cdot(k-1)+2}] \pmod{2^b - 1} = \sum_{i=1}^{k} C_i \cdot \hat{B}_{4\cdot(i-1)+2} \pmod{2^b - 1} \tag{15}$$

- *Core 3*

$$C_{\hat{B}3} = [C_1 \cdot \hat{B}_3 + C_2 \cdot \hat{B}_7 + \cdots + C_k \cdot \hat{B}_{4\cdot(k-1)+3}] \pmod{2^b - 1} = \sum_{i=1}^{k} C_i \cdot \hat{B}_{4\cdot(i-1)+3} \pmod{2^b - 1} \tag{16}$$

- *Core 4*

$$C_{\hat{B}4} = [C_1 \cdot \hat{B}_4 + C_2 \cdot \hat{B}_8 + \cdots + C_k \cdot \hat{B}_{4\cdot(k-1)+4}] \pmod{2^b - 1} = \sum_{i=1}^{k} C_i \cdot \hat{B}_{4\cdot(i-1)+4} \pmod{2^b - 1} \tag{17}$$

If we add to this $K/128$ shift operations (K - the number of data bits) we easily calculate that each processing core requires $T_1 = 8 \cdot k + K/128 - 1$ clock cycles (k accesses to the L1 cache, k integer multiplications, $k - 1$ integer additions and $K/128$ shift operations) to compute the value of the check-byte $C_{\hat{B}P}$ (P = 1, 2, 3, 4). After finishing this task, each core will take $T_2 = 2$ clock cycles (1 integer subtraction and 1 modulo reduction) to perform the following operations:

- *Core 1*

$$S_1 = [C_{\hat{B}1} - \hat{C}_{B1}] \pmod{2^{32}-1} \tag{18}$$

- *Core 2*

$$S_2 = [C_{\hat{B}2} - \hat{C}_{B2}] \pmod{2^{32}-1} \tag{19}$$

- *Core 3*

$$S_3 = [C_{\hat{B}3} - \hat{C}_{B3}] \pmod{2^{32}-1} \tag{20}$$

- *Core 4*

$$S_4 = [C_{\hat{B}4} - \hat{C}_{B4}] \pmod{2^{32}-1} \tag{21}$$

As explained in Section 2, if the data are received in error ($S \neq 0$), the decoder will additionally perform n_{TL} table lookups, n_{TL} comparisons, 1 integer addition and 1 modulo reduction. In our case, four such operations will be executed in parallel in $T_3 = 29 \cdot n_{TL} + 2$ clock cycles. Consequently, if we sum up all processing times, we come to the conclusion that the processor requires

$$T_{total} = T_1 + T_2 + T_3 = 8 \cdot k + K/128 + 29 \cdot n_{TL} + 3 \tag{22}$$

clock cycles to process $4 \cdot K$ data bits, i.e. one second to decode

$$G = \frac{C_R}{T_{total}/(4 \cdot K)} = \frac{(3.5 \cdot 10^9) \cdot 4 \cdot K}{8 \cdot k + K/128 + 29 \cdot n_{TL} + 3} \tag{23}$$

data bits. By substituting the values of K, k and n_{TL} in (23), we obtain that G_{min} = 16.93 Gbps and G_{max} = 34.38 Gbps. In other words, all codes from Table 4 have the potential to be used in 10G networks (e.g. 10G PONs) [1]. In addition, from Table 4 we see that the codes with code rate 0.9922 have theoretical throughputs above 33 Gbps. Thus, they could be candidates for use in ONWOAs operating at 32 Gbps (e.g. 32G Fibre Channel network) [1]. Finally, from (14)-(21) it is evident that the analyzed codes (Table 4) can correct burst/random asymmetric errors affecting four adjacent 32-bit bytes. This feature makes them more attractive for potential use, since in practice channel errors may corrupt two adjacent bytes [4], [5], [6], [7].

Table 4. Theoretical Decoding Throughputs for Some Four-Byte Interleaved Codes.

Four-Byte Interleaved $(B_{l/32}AEC/R_3AEC)_{32}$ Code	l	Code Rate	Decoding Throughput
(1056, 1024)	8	0.9697	16.93 Gbps
(1056, 1024)	9	0.9697	16.93 Gbps
(2080, 2048)	8	0.9846	25.81 Gbps
(2080, 2048)	9	0.9846	25.15 Gbps
(4128, 4096)	8	0.9922	34.38 Gbps
(4128, 4096)	9	0.9922	33.79 Gbps

4. Comparison with Existing Codes of Similar Properties

In Section 1 we listed several codes capable of correcting multiple asymmetric errors within a b-bit byte. Among them, the most interesting are codes presented in [10] and [13]. These codes have similar error correction properties as the proposed codes, and hence, they will be compared to each other.

At the outset, let us focus on the parameters of all codes. According to [10], the codes correcting single b-bit byte asymmetric errors (the S_bAEC codes) have $b + \log_2(k)$ check bits, where $k < 2^{b-1}$. This means that the S_{16}AEC codes require up to 23 check-bits to protect data words of length $K \leq 2048$ bits. On the other hand, although the presented codes always have b check bits, they cannot be constructed when $l = 8$, $t = 3$ and $b = 16$. Hence, to protect data words of length $K \leq 2048$ bits, one must use integer $(B_8AEC/R_3AEC)_b$ codes that have 32 check-bits. The same applies to codes correcting t asymmetric errors within a b-bit [13]. For the parameters $t = 3$ and $b = 16$, these codes can protect data words of length up to 224 bits. However, for the parameters $t = 3$ and $b = 32$ they can protect more than 2048 bits.

As far as data processing is concerned, the codes from [10] are significantly different from the proposed codes and the codes given in [13]. Namely, besides using integer and LUT operations, these codes also use bit-oriented operations (b - 1 binary additions per b-bit byte [10]). Because of this, they are not suited for implementation on modern processors. On the other hand, in Section 3 we have seen that the proposed codes are very suitable for software implementation. The same holds for the codes presented in [13]. The only difference between these two codes is that codes from [13] are less demanding in terms of memory needs. More precisely, for data words of length $K \leq 2048$ bits they require up to 3.17 MB of memory. In contrast to them, the proposed codes demand at most 6.29 MB of memory (Table 5).

Table 5. Comparison of the Proposed Codes with Existing Codes of Similar Properties.

Main characteristics	Proposed codes	Codes from [13]	Codes from [10]
Error correction capabilities	Correction of burst and random asymmetric errors within a b-bit byte	Correction of random asymmetric errors within a b-bit byte	Correction of any asymmetric error within a b-bit byte
Maximum number of data bytes	Not specified (depends on the results of a computer search)	Not specified (depends on the results of a computer search)	The largest prime less than 2^k
Number of check-bits	b	b	$b + \log_2(k)$
Processing of data bits	Integer and LUT operations	Integer and LUT operations	Integer, LUT and bit-oriented operations
Size of the syndrome table when $K \leq 2048$ bits	≤ 6.29 MB	≤ 3.17 MB	≤ 64 B

Finally, it must be mentioned that the proposed codes provide the same level of reliability as S_bAEC codes. Namely, although they cannot correct any asymmetric error within a *b*-bit byte, they are sufficiently powerful to correct almost all channel errors. Such a conclusion is indirectly supported by experimental results reported in [4], [5], [6], [7]. On the other hand, the codes from [13] are unable to correct burst asymmetric errors affecting four or more bits. Because of this, they cannot provide high throughput and reliability simultaneously.

5. Conclusion

In this paper we presented a class of codes suitable for use in optical networks without optical amplifiers. We have shown that these codes have two important characteristics: first, they can correct burst and random asymmetric errors confined to a *b*-bit byte, and second, they use integer and lookup table operations which make them suitable for software implementation. In addition to this, it was also shown that the proposed codes can be interleaved without delay and without using any additional hardware. Thanks to this fact, it is possible to construct simple codes capable of correcting multiple asymmetric errors affecting several consecutive bytes.

References

[1] R. Ramaswani, K. Sivarajan and G. Sasaki, *Optical Networks: A Practical Perspective*, 3rd ed., Elsevier, Inc., 2010.

[2] J. R. Pierce, "Optical Channels: Practical Limits with Photon Counting," *IEEE Trans. Communications*, vol. 26, pp. 1819-1821, Dec. 1978.

[3] P. Oprisan and B. Bose, "ARQ in Optical Networks," *Proc. IEEE Int'l Symp. Pacific Rim Dependable Computing*, pp. 251-257, Dec. 2001.

[4] D. Mello, E. Offer and J. Reichert, "Error Arrival Statistics for FEC Design in Four-Wave Mixing Limited Systems," *Proc. Optical Fiber Communications Conference*, pp. 529-530, Mar. 2003.

[5] L. James, "Error Behaviour in Optical Networks," PhD thesis, Department of Engineering, University of Cambridge, 2005.

[6] P. Anslow and O. Ishida, "Error Distribution in Optical Links," *IEEE 802.3 HSSG Interim Meeting*, Nov. 2007.

[7] K. Cho *et al.*, "Performance of Forward-Error Correction Code in 10-Gb/s RSOA-Based WDM PON," *IEEE Photon. Technology Letters*, vol. 22, no. 1, pp. 57-59, Jan. 2010.

[8] S. Park and B. Bose, "Burst Asymmetric/Unidirectional Error Correcting/Detecting Codes," *Proc. IEEE 20th Int'l Symp. Fault-Tolerant Computing*, pp. 273-280, June 1990.

[9] Y. Saitoh and H. Imai, "Some Classes of Burst Asymmetric or Unidirectional Error Correcting Codes," *Elect. Letters*, vo1. 26, no. 5, pp. 286-287, Mar. 1990.

[10] B. Bose and S. Al-Bassam, "Byte Unidirectional Error Correcting and Detecting Codes," *IEEE Trans. Computers*, vol. 41, no. 12, pp. 1601-1606, Dec. 1992.

[11] S. Al-Bassam and B. Bose, "Asymmetric/Unidirectional Error Correcting and Detecting Codes," *IEEE Trans. Computers*, vol. 43, no. 5, pp. 590-597, May 1994.

[12] A. Radonjic and V. Vujicic, “Integer Codes Correcting Burst Errors within a Byte,” *IEEE Trans. Computers*, vol. 62, no. 2, pp. 411-415, Feb. 2013.

[13] A. Radonjic and V. Vujicic, “Integer Codes Correcting Spotty Byte Asymmetric Errors,” *IEEE Comm. Letters*, vol. 20, no. 12, pp. 2338-2341, Dec. 2016.

[14] M. Ruiz-Sanchez, E. Biersack and W. Dabbous, “Survey and Taxonomy of IP Address Lookup Algorithms,” *IEEE/ACM Trans. Networking*, vol. 15, no. pp. 8-23, Mar./Apr. 2001.

[15] K. Mehlhornand and P. Sanders, *Algorithms and Data Structures: The Basic Toolbox*, Springer, 2008.

[16] R. Giladi, *Network Processors: Architecture, Programming, and Implementation*, Elsevier, Inc., 2008.

[17] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Elsevier, Inc., 2012.

[18] A. Fog, “The Microarchitecture of Intel, AMD and VIA CPUs: An Optimization Guide for Assembly Programmers and Compiler Makers,” Tech. University of Denmark, Feb. 2017.

Appendix A

First 128 Coefficients C_l in $[2, 2^{32} - 2]$ for Integer $(B_l AEC/R_t AEC)_{32}$ Codes

$l = 8$ and $t = 2$															
2	259	261	263	265	269	271	277	281	283	289	293	299	307	311	313
317	337	341	347	349	353	359	361	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463	467	479	487	491	499	503
509	521	523	541	547	557	563	569	571	577	587	593	599	601	607	613
617	619	631	641	643	647	653	659	661	673	677	691	701	709	719	727
733	739	743	751	757	761	773	787	797	809	811	821	823	827	829	839
853	857	859	863	877	881	883	887	907	911	919	929	937	941	947	953
967	971	977	983	991	997	1009	1013	1019	1021	1039	1049	1051	1061	1063	1069
$l = 8$ and $t = 3$															
2	263	267	269	271	275	281	283	287	299	307	313	317	349	353	361
367	373	379	383	389	397	401	409	421	431	433	443	463	467	479	487
499	503	509	523	527	541	547	557	563	587	593	599	601	613	617	619
631	643	647	653	659	661	673	677	691	701	727	733	739	743	751	757
761	773	787	797	809	811	823	827	831	839	841	853	857	859	863	873
877	879	881	883	887	907	911	919	929	937	947	953	967	971	977	983
991	997	1009	1011	1013	1019	1021	1031	1039	1049	1051	1061	1063	1069	1077	1087
1091	1093	1097	1103	1109	1117	1123	1129	1151	1163	1181	1187	1193	1201	1213	1217
$l = 9$ and $t = 2$															
2	515	517	519	521	523	527	529	533	541	547	551	553	557	563	569
571	577	587	593	599	601	607	613	617	619	631	643	647	653	659	661
673	677	691	701	709	719	727	733	739	743	751	757	761	769	773	787
797	809	811	821	823	827	829	839	853	857	859	863	877	881	883	887
907	911	919	929	937	941	947	953	967	971	977	983	991	1009	1013	1019
1021	1031	1039	1049	1051	1061	1063	1069	1087	1091	1093	1097	1103	1109	1117	1123
1129	1151	1163	1171	1181	1187	1193	1201	1213	1217	1223	1229	1231	1237	1249	1259
1277	1279	1289	1291	1297	1301	1303	1307	1319	1321	1327	1361	1367	1369	1373	1381
$l = 9$ and $t = 3$															
2	519	523	527	533	541	547	553	557	563	575	583	587	593	599	601
613	617	619	631	643	647	653	659	661	673	677	691	701	703	727	733
739	743	751	761	773	787	797	809	811	823	827	839	841	853	857	859
863	877	881	883	887	907	911	919	929	937	947	953	967	971	977	983
991	1009	1013	1019	1021	1031	1039	1049	1051	1061	1063	1069	1091	1093	1097	1103
1109	117	1123	1129	1151	1163	1181	1187	1193	1201	1213	1217	1223	1231	1237	1249
1259	1277	1279	1289	1291	1297	1301	1303	1307	1319	1327	1361	1373	1381	1399	1423
1427	1429	1433	1439	1453	1459	1471	1481	1483	1487	1489	1493	1499	1523	1531	1543

APPENDIX B

MATLAB CODE FOR FINDING THE COEFFICIENTS C_I

```
function [ ] = Coefficients

b=input('\nPlease, specify the byte length (in bits): \nb = ');
l=input('\nPlease, specify the parameter l (l < b): \nl = ');
t=input('\nPlease, specify the parameter t (t=2 or t=3): \nt = ');
if (l>b)||(t>3)||(t<2)||(t>=l)
    fprintf('Error! The following condition must be satisfied: 2 <= t <=3 < l < b)\n');
    return;
end
Q=2^b-1;
q1=0;
s1=[ ];
for i=0:b-1
      for k=1:2^l-1
            q1=q1+1;
            s1 (q1)=mod(k*(2^i),Q);
      end
end
f3=[ ];
q3=0;
for i=0:b-3
      for j=i+1:b-2
            for k=j+1:b-1
                  q3=q3+1;
                  f3(q3)=mod(2^i+2^j+2^k,Q);
            end
      end
end
f2=[ ];
q2=0;
for i=0:b-2
      for j=i+1:b-1
            q2=q2+1;
            f2(q2)=mod(2^i+2^j,Q);
      end
end
if (t==2)
     e=[s1 f2];
elseif (t==3)
     e=[s1 f2 f3];
end
X=setdiff(e,0);
L=length(X);
E=X;
Coefficients=[-1];
k=1;
for m=2:Q-1
      A=[ ];
      z=0;
      for i=1:L
            A(i)=mod(-m*X(i),Q);
      end
      if (nnz(setdiff(A,E))==L)
           k=k+1;
           if (k<=129)
               Coefficients(k)=m;
               E=union(E,A);
           else
               break;
           end
      end
end
Ci=setdiff(Coefficients,-1);
fprintf(1,'The following coefficients are found: %d\n',Ci);
```